AA Megoldó Arnold Aladár AA

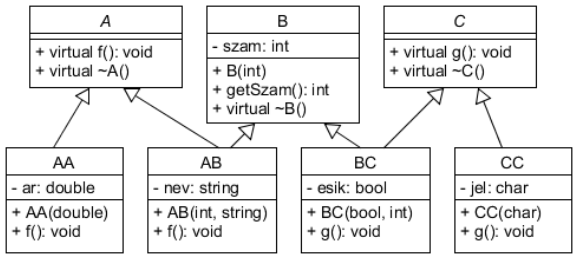
Programozás alapjai 2. (inf.) 2. ZH 2016.05.05. gyak./lab. hiányzás: 0/0+0	G5-QB309/L4-R4N
ABC123 **Aud.Max/1.**	kZH: 6.5+7

f.	max.	elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	**40**		

Minden beadandó megoldását a feladatlapra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlapra írt megoldásokat értékeljük! *A megoldások során feltételezheti, hogy minden szükséges input adat az előírt formátumban rendelkezésre áll. A feladatok megoldásához csak a letölthető C, C++ és STL összefoglaló használható. Elektronikus eszköz (pl. tablet, notebook, mobiltelefon) nem használható. A feladatokat* ***figyelmesen olvassa*** *el, megoldásukhoz csak akkor használjon STL tárolót, ha a feladat ezt külön engedi/kéri!*
Ne írjon felesleges *függvényeket ill. kódot! A feleletválasztós feladatoknál a hibás válaszért pontlevonás jár. A részfeladatokra kapott pontok a feladaton belül előjelesen összeadódnak. negatív összeg esetén a feladatra kapott pontszám 0. Az első feladatrészben (beugró) minimum 5 pontot el kell érnie ahhoz, hogy a többit értékeljük.*

1. feladat: Beugró. A feladatok megoldásához **használjon STL elemeket!** **Σ 10 pont**

a) Deklarálja az alábbi osztálydiagramon szereplő osztályok közül az AB osztályt és a vele közvetlen kapcsolatban állókat! A konstruktorokat valósítsa is meg! A tagváltozók kezdeti értékét a konstruktorparaméterek adják. (3p)




```
struct A {
   virtual void f() = 0;
   virtual ~A();
};

class B {
   int szam;
public:
   B(int i) :szam(i) {}
   int getSzam() const;
   virtual ~B();
};

class AB : public A, public B {
   string nev;
public:
   AB(int i, string, s) :B(i), nev(s) {}
   void f();
}
```

b) Mikor beszélünk a C++ nyelv kapcsán roll back-ről? Mit jelent? (1p)
Kivételkezeléskor a try blokkban **automatikusan keletkező** objektumok **megszüntetésének** folyamata a roll back vagy stack unwinding.

c) Az a) részfeladat összes osztálya rendelkezésére áll! Az *std::vector* sablon felhasználásával hozzon létre egy olyan objektumot (*tar*), amivel „tárolni” tud *AA* és *AB* típusú objektumpéldányokat is. Ezután a dinamikus memóriában hozzon létre 100 db AB példányt és egy AA példányt úgy, hogy később elérje azokat! Ügyeljen a konstruktorok paraméterezésére! (2p)

```
std::vector<A*> tar;
for (int i = 0; i < 100; i++)
    tar.push_back(new AB(1, "C++"));
tar.push_back(new AA(3.14));
```

d) Mit ír ki az alábbi programrészlet a kimenetre? (1p)

```
struct A {
  A() { f(); }
  virtual void f() { std::cout << "A::f";}
};
struct B : public A {
  void f() { std::cout << "B::f";}
};
int main { B b; return 0; }
```

Kiírás: A::f

e) Mi a hiba az alábbi programrészletben? (1p)

```
   int *ip = new int[10];
   delete[10] ip;
```

delelete[10] helyett delete[] kell.

f) Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

Referencia típusú tagváltozót konstruktor törzsében lehet inicializálni.	I	H
Absztrakt osztálynak kell, hogy legyen virtuális tagfüggvénye.	I	H
Minden osztályból létre lehet hozni tömböt.	I	H
Privát (`private`) adattagot a származtatott objektumból mindig el lehet érni közvetlenül.	I	H

2. Feladat **Σ 10 pont**

a) Készítsen adapter sablont (*Indexelheto*), ami minden indexelhető (*operator[]*) szabványos sorozattárolóra alkalmazható és segítségével egy M elemet tartalmazó tároló első eleme N, a második elem N+1, harmadik N+2 …N+M-1 indexértékkel érhető el, ahol N, és M tetszőleges egész. Alapértelmezésként N = 0, a tároló pedig az *std::vector* legyen! A sorozattároló minden tagfüggvénye legyen elérhető, kivéve az *at()*! Ügyeljen a sorozattárolókra jellemző konstruktorok megvalósítására is!
Példa a használatra: (4p)

```
Indexelheto<int, 10> v10(2);    // 10-től indexelhető 2 elemű vektor
v10[10] = 1;                    // első eleme 1
v10[11] = 2;                    // második eleme 2;
```

b) Hozzon létre az elkészített adapter és az *std::deque* felhasználásával egy 20-tól indexelhető 20 elemű egész tömböt! (1p)

c) Írjon C++ függvénysablont (*keres*), ami egy iterátorokkal megadott adatsorozatban megkeresi az első olyan elemet, amire az adott predikátum igaz értéket ad! A függvény első két paramétere két iterátor, amivel a szokásos módon megadjuk a jobbról nyílt intervallumot. A függvény 3. paramétere pedig egy predikátum, ami egy egyparaméteres függvény vagy függvényobjektum. Amennyiben nincs a feltételnek megfelelő elem, akkor az adatsorozat végét jelző értékkel (iterátor) térjen vissza a függvény! Ha jól oldja meg a feladatot, akkor az alábbi kódrészlet lefutása után az eredmény a 3-as indexű elemre (-16) fog mutatni. (2p)

```
bool negativ(int a) { return a < 0; }
int sorozat[] = { 1, 4, 9, -16, 25, 0, 72, 100, 0};      // a sorozat
int *eredmeny = keres(sorozat, sorozat+9, negativ);
```

d) Készítsen olyan függvényobjektum sablont, ami a keres sablonnal felhasználható az olyan elemek megkeresésére, amelyek nagyobbak a függvényobjektum konstruktorában megadott értéknél! (2p)

e) A részfeladatok eredményeit felhasználva írjon kódrészletet, ami a **b)** részfeladatban létrehozott tömbből kiírja a szabványos kimenetre az első 26-nál nagyobb értéket! (Feltételezheti, hogy van ilyen.) (1p)

Egy lehetséges megoldás:

```
template <typename T, size_t N = 0, class C = std::vector<T> >
class Indexelheto : public C {
    T& at(size_t ix);
    T at(size_t ix) const;
public:
    Indexelheto(size_t n = 0, const T& value = T()) : C(n, value) {}
    template<typename Iter>
    Indexelheto(Iter first, Iter last) : C(first, last) {}
    T& operator[](size_t ix) { return C::operator[](ix-N); }
    T operator[](size_t ix) const { return C::operator[](ix-N); }
};

Indexelheto<int, 20, std::deque<int> > d20(20);

template <class Iter, class P>
Iter keres(Iter first, Iter last, P pred) {
    while (first != last){
        if (pred(*first)) return first;
        ++first;
    }
    return first;
}

template <class T>
class NagyobbMint {
    T ref;
public:
    NagyobbMint(const T& a) : ref(a) {}
    bool operator()(const T& a) const { return a > ref; }
}

std::cout << *keres(d20.begin(), d20.end(), NagyobbMint<int>(26));
```

3. Feladat **Σ 10 pont**

A *Diak* osztályban diákok adatait tároljuk.

```
class Diak {
    std::string nev;
    double atlag;
public:
    Diak(const std::string& n = "", double a = 0);
    double getAtlag() const;
    std::string getNev() const;
    void setAtlag(double a);
    void setNev(const std::string& n);
    virtual ~Diak();
};
```

Adott továbbá a *Serializable* osztály.

```
struct Serializable {
    virtual void write(ostream& os) const = 0;
    virtual void read(istream& is) = 0;
    virtual ~Serializable() {}
};
```

a) A fenti osztályok felhasználásával, de azok módosítása nélkül hozzon létre a *Diak* osztállyal kompatibilis, perzisztens *PDiak* osztályt! Megoldásában vegye figyelembe, hogy a névben szóköz is lehet! Az elmentett állapot visszatöltésekor az osztály végezzen ellenőrzést, hogy jó adatokat kap-e, azonban nem kell bombabiztos megoldás! Hiba esetén dobjon *std::out_of_range* kivételt! 5p

b) Egy rövid kódrészletben hozzon létre egy *PDiak* példányt a saját nevével. Átlagnak 5-öt adjon meg! Mentse ki az objektum adatait a szabványos kimenetre! Kommentben adja meg, hogy mit írt ki! Jelölje a nem látható karaktereket is! 2p

c) Tételezze fel, hogy rendelkezésére áll a gyakorlaton elkészített *PKomplex* osztály is, ami szintén a *Serializable* osztály segítségével valósítja meg a perzisztens viselkedést! Egészítse ki megoldását, hogy az alábbi kódrészlet az elvárásoknak megfelelően működjön, de minden más *Serializable* leszármazottal is használható legyen! 3p

```
PDiak d1("Nagy Elemer", 5.0), d2("Kiss Szaniszlo", 4.0);
PKomplex k1(2, 4);
stringstream ss;
d1.write(ss);
k1.write(ss);
d2.write(ss);
PDiak nd1, nd2;
PKomplex nk1;
ss >> nd1 >> nk1 >> nd2;      // elvárjuk, hogy d1 == nd1 && d2 == nd2 && k1 == nk1 legyen,
                              // azaz az elmentett állapot álljon elő.
```

Egy lehetséges megoldás:

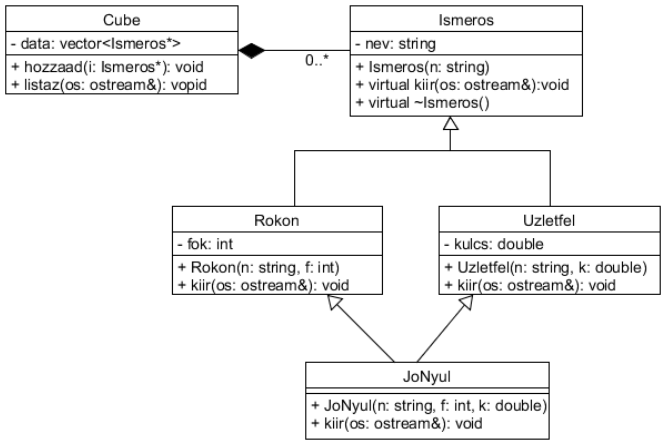
```
struct PDiak : public Diak, public Serializable {
    PDiak(const std::string& n = "", double a = 0) : Diak(n,a) {}
    void write(ostream& os) const {
        os << "PDiak" << endl;
        os << getNev() << endl;
        os << getAtlag() << endl;
    }
    void read(istream& is) {
        string line;
        (is >> line).ignore(1);
        if (line != "PDiak") throw out_of_range("PDiak read");
        getline(is, line);
        setNev(line);
        double a;
        (is >> a).ignore(1);
        setAtlag(a);
    }
};
PDiak d1("Nagy Elemer", 5.0);
d1.write(std::cout); // PDiak\nNagy Elemer\n5.0\n
istream& operator>>(istream& is, Serializable& s) {
        s.read(is);
        return is;                                          0,5 pont
}
```

4. Feladat **Σ 10 pont**

A Százholdas Pagonyban Nyuszi informatikai rendszerben (*Cube*) szeretné nyilvántartani az ismerőseit (*Ismeros*). Első körben csak a rokonait (*Rokon*) és üzletfeleit (*Uzletfel*), de később a rendszert bővíteni akarja, hogy a barátok és ellenségek adatai is benne legyenek. Azt azonban már most is tudja, hogy van olyan rokona, aki egyben üzletfele is (*JoNyul*, pl. akivel az erdei kisvasutat is építteti). Mindenkinek kiíratható (*kiir*) a neve (*string*), a rokonoknak a rokonsági foka (*int*), az üzletfeleknek pedig az adókulcsa (*double*) is. A *JoNyul*aknak neve, rokonsági foka és adókulcsa is van, és kezelhetők rokonként és üzletfélként is. Kiíratáskor minden adatuk kiíródik (nem baj, ha többször is). Az adatok kiíratása a függvényparaméterként megadott adatfolyamra (*std::ostream)* történik. A Cube rendszerbe új ismerőst bevinni a *hozzaad* metódussal lehet, illetve paraméterben megadott adatfolyamra ki lehet listázni az összes ismerős minden adatát (*listaz*). Ha a Cube rendszer megsemmisül, a benne tárolt adatok is elvesznek.

Tervezzen objektum-orientált megoldást a fenti leírás alapján a dőlt betűs megnevezések felhasználásával! Az attribútumok kivétel nélkül legyenek privát elérésűek és konstruktorban állíthatók. Rajzoljon UML osztálydiagramot, amin szerepeljenek az attribútumok és a metódusok. UML jelöléssel jelölje a láthatóságot is.
Definiálja a **Cube**, **Ismeros**, **Rokon**, és **JoNyul** osztályokat. A konstruktorokat *inline* módon adja meg, a többi metódust csak deklarálja. Másoló konstruktort nem kell készíteni. Használjon STL tárolót és algoritmust!
Valósítsa meg a Cube osztály **hozzaad** és **listaz** metódusát, és az ezek végrehajtása közben meghívott további metódusokat minden definiált osztályhoz!

Egy lehetséges megoldás:




```
class Ismeros {
        string nev;
public:
        Ismeros(string n) : nev(n) {}
        virtual void kiir(ostream&) const;
        virtual ~Ismeros();
};
class Cube {
        vector<Ismeros*> data;
public:
        void hozzaad(Ismeros* i);
        void listaz(ostream&) const;
        ~Cube();
};
```

```
class Rokon : virtual public Ismeros {
        int fok;
public:
        Rokon(string n, int f) : Ismeros(n), fok(f)
{}
        virtual void kiir(ostream& os) const;
};
class JoNyul: public Rokon, public Uzletfel {
public:
        JoNyul(string n, int f, double a) :
Ismeros(n), Rokon(n,f), Uzletfel(n,a) {}
        virtual void kiir(ostream& os) const;
};
```

```
void Cube::listaz(ostream& os) {
        for (int i = 0; i < data.size(); i++) data[i]-
>kiir(os);
}
void Ismeros::kiir(ostream& os) {
        os << nev;
}
void Rokon::kiir(ostream& os) {
        Ismeros::kiir(os);
        os << " " << fok;
}
void JoNyul::kiir(ostream& os) {
        Rokon::kiir(os);
        os << " ";
        Uzletfel::kiir(os);
}
```